OOPs Abstraction

Mohammed Husain Bohara

So far we have covered

1. Class
2. Object
3. Inheritance
4. Polymorphism

What is abstraction in Java?

Abstraction is a process of **hiding** the implementation details and showing only functionality to the user.

Hiding the process, highlighting set of services

Any real time example of abstraction?

Bank ATM

What is the difference between Abstraction & Encapsulation?

How many ways to achieve abstraction?

There are two ways to achieve abstraction in java

- Abstract class (0 to 100%)
- Interface (100%)

Abstract class

- A class that is declared with **abstract** keyword, is known as abstract class in java.
- It can have abstract and non-abstract methods (method with body).

We will discuss all 5 element for abstract class

1. Instance variable possible
2. Method can be abstract
3. Constructor is possible for abstract class
4. Instance block is also possible
5. Static block is possible too

Interview questions?

What is the need of abstract class?

How to prevent the object creation in java?

To prevent object creation

1.
Abstract class instance variable

We will discuss it with the abstract class constructor

2.
Abstract class method

```
//Abstract class
            abstract class A{}

//Abstract method
            abstract void printStatus();
```

How normal method different from abstract method?

1. Abstract method contain only method declaration
2. Abstract method end with semicolon
3. To represent abstract method use abstract keyword

If class contain at least one abstract method then class called abstract class

```
class Test
{
        void m1() {      }
        abstract void m2();
}
```

```
abstract class Test
{
        void m1() {      }
        abstract void m2();
}
```

What is the difference between normal class & abstract class?

It is not possible to create an object of abstract class but we can create an object for normal class.

This is not a complete definition of abstract class.

Example

```
abstract class Test
{
        void m1() {       }
        abstract void m2();
}
```

```
abstract class Test
{
        void m1() {       }
        void m2(){       }
}
```

Complete definition of abstract class

1. Abstract class may contain abstract method
2. Abstract class may not contain abstract method
3. It is not possible to create abstract class object

Example-1

```
abstract class Parent
{
abstract void m1();
abstract void m2();
void m3()
{  S.O.P.("m3 method");
}
}
```

NO

```
class Child extends Parent
{
 void m1()
  { S.O.P.("m1 method");}
 void m2()
 { S.O.P.("m2 method");}
P.S.V.M()
{
Parent p = new Parent();
Child c = new Child();
}
}
```

Will it compile successfully?

Example-2

```
abstract class Parent
{
abstract void m1();
abstract void m2();
void m3()
{  S.O.P.("m3 method");
}
}
```

Yes

```
class Child extends Parent
{
 void m1()
  { S.O.P.("m1 method");}
 void m2()
 { S.O.P.("m2 method");}
P.S.V.M()
{
Parent p = new Child();
p.m1(); p.m2(); p.m3()
}
}
```

Will it compile successfully?

Example-3

```
abstract class A
{
abstract void m1();
abstract void m2();
abstract void m3();
}
class B extends A
{
 void m1()
  { S.O.P.("m1 method");}
}
```

```
class C extends B
{
void m2()
 { S.O.P.("m2 method");}
}
```

```
class D extends C
{
void m3()
 { S.O.P.("m3 method");}
P.S.V.M()
{
D d = new D();
d.m1(); d.m2(); d.m3();
}
```

Will it compile successfully?

NO

What changes we required to make previous code executable?

Declare class as abstract class

Why?

Example-3

```
abstract class A
{
abstract void m1();
abstract void m2();
abstract void m3();
}
abstract class B extends A
{
 void m1()
  { S.O.P.("m1 method");}
}
```

```
abstract class C extends B
{
void m2()
 { S.O.P.("m2 method");}
}
class D extends C
{
void m3()
 { S.O.P.("m3 method");}
P.S.V.M()
{
D d = new D();
d.m1(); d.m2(); d.m3();
}
```

YES

Will it compile successfully?

Is it possible to declare main method in abstract class?

```
abstract class Test
{
        P.S.V.M()
        {
                S.O.P(“Hello abstract class”);
        }
}
```

Compile & run successfully

3.
Abstract class constructor

Can abstract class have constructor?

Inside the abstract class, constructor declaration is possible or not?

```
abstract class Test
{
        Test()
        {
        }
        P.S.V.M()
        {
                S.O.P("Hello abstract class");
        }
}
```

It's possible

Inside the abstract class, constructor declaration is possible or not?

```
abstract class Test
{
        Test()
        {
        }
        P.S.V.M()
        {
                S.O.P("Hello abstract class");
                Test t = new Test();
        }
}
```

It will generate an error- object creation is not allowed

How to execute the abstract class constructor without object creation?

```
abstract class Test
{
Test()
{ S.O.P.("abstract class
constructor");
}
abstract void m1();
}
```

```
class  Test1 extends Test
{
void m1()
{
 S.O.P.("m1 method");
}
Test1()
{
  super();
  S.O.P.("normal class constructor");
}
P.S.V.M()
{
Test1 t = new Test1();
t.m1();
}
}
```

Conclusion

For abstract class constructor declaration is allowed & possible to execute constructor also

Abstract class instance variable example

```
abstract class Parent
{
int x;
Parent(int x)
{ this.x=x;
}
abstract void details();
}
```

```
class  Child extends Parent
{
int y;
Child(int x, int y)
{
  super(x);
  this.y=y;
}
void details()
{
 S.O.P.(“Parent x: ” + super.x);
 S.O.P.(“Child y: ” + this.y);
}
}
```

```
class Test{
P.S.V.M()
{
Child c = new Child(10,20);
c.details();
}}
```

4.
Instance & static block for abstract class

Example

```
abstract class Test
{
{
S.O.P.("abstract class instance block");
}
static
{
S.O.P.("abstract static block");
}
}
```

```
class  Test1 extends Test
{
P.S.V.M()
{
Test1 t = new Test1();
}
}
```

Compile & execute successfully

Can we inherit abstract class to another abstract class?

one abstract class **extends** another abstract class

Yes you can

Example

```
abstract class One {

 }

abstract class Two extends One {

}
```

Duplicate abstract method Example

```
abstract class One {
abstract void m1();
void m1(){
   System.out.println("non abstract method" );
} }
```

```
public class MyClass extends One {
void m1()
{     System.out.println("abstract class method" );
}
public static void main(String args[]) {
            System.out.println("My class");
            MyClass m = new MyClass();
            m.m1();
}}
```

Error-

method m1() is already defined in class One

Will it compile or not?

What is the output here

```
abstract class Bike{
  abstract void run();
}

class Honda extends Bike{
    void run()
        {
             System.out.println("running safely..");
        }
         public static void main(String args[]){
              Bike obj = new Honda();  // is there any error here
               obj.run();
              }
}
```

No error
Execute
successfully

What is the output here

```
abstract class Bank{
        abstract int getRateOfInterest();
}
class SBI extends Bank{
        int getRateOfInterest(){return 7;}
}
class PNB extends Bank{
        int getRateOfInterest(){return 8;}
}

class TestBank{
        public static void main(String args[]){
                Bank b;
                b=new SBI();
       System.out.println("Rate of Interest is: "+b.getRateOfInterest()+" %");
                b=new PNB();
      System.out.println("Rate of Interest is: "+b.getRateOfInterest()+" %");
}}
```

What is the output here

```
abstract class Bike{
            Bike()    { System.out.println("bike is created");}
            abstract void run();
            void changeGear()   { System.out.println("gear changed");}
 }

 class Honda extends Bike{
            void run()  { System.out.println("running safely..");}
 }
 class TestAbstraction{
            public static void main(String args[]){
            Bike obj = new Honda();
            obj.run();
            obj.changeGear();  // is it work here?
             }
}
```

YES

Very Important concept

make an array of the abstract class

```
public abstract class Game{
  ...
}

Game games = new Game(); //Error
Game[] gamesArray = new Game[10]; //No Error
```

Conclusion

Instantiation means creation of an instance of a class.

```
Game[] gamesArray = new Game[10];
```

In this scenario, you've just declared a gamesArray of type Game with the size 10(just the references and nothing else). That's why its not throwing any error.

What is Interface in java?

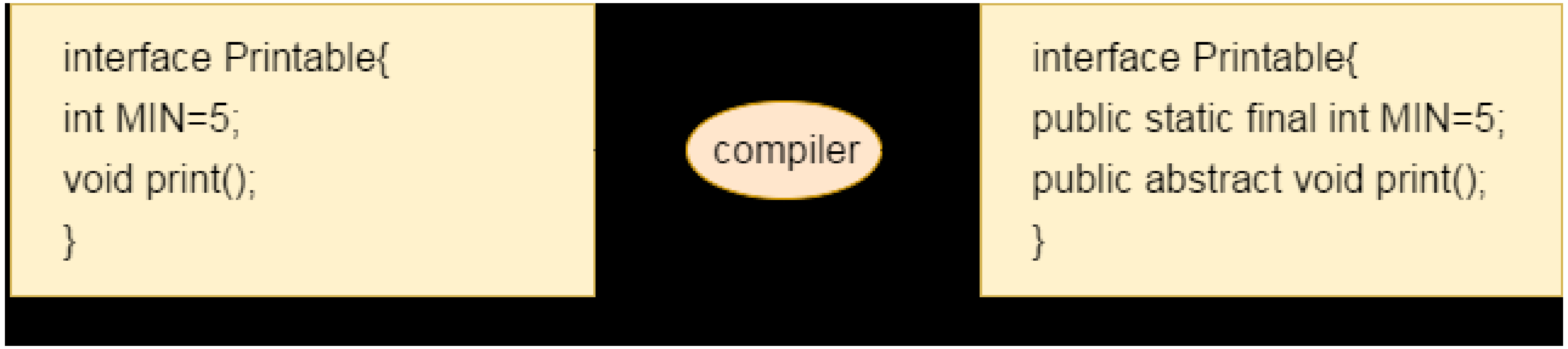
- The interface in java is **a mechanism to achieve abstraction**. There can be only abstract methods in the java interface not method body. It is used to achieve abstraction and multiple inheritance in Java.
- Java Interface also **represents IS-A relationship**.
- It cannot be instantiated just like abstract class.
- Interface is also known as 100% pure abstract class

What is the purpose of Interface?

- It is used to achieve abstraction.
- By interface, we can support the functionality of multiple inheritance.
- It can be used to achieve loose coupling.

How do we use interface in program?

```
interface Player
{
    int id = 10;
    int move();
}
```



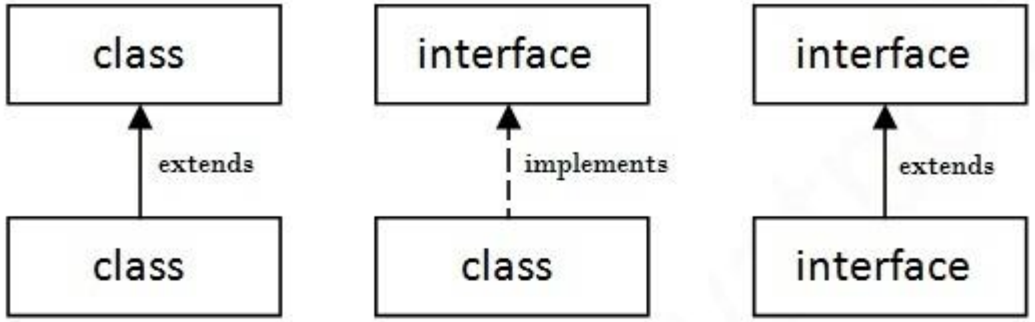
```
interface Printable{
int MIN=5;
void print();
}
```

```
interface Printable{
public static final int MIN=5;
public abstract void print();
}
```

What is the conclusion here?

1. Interface are by default abstract
2. Variables are by default public, static & final
3. Methods are by default public & abstract

How to create relationship between classes and interfaces?



Will it compile or not?

```
Interface Test{
          void m1();
          void m2();
          void m3();
}
Class Test1 implements Test{
          void m1(){     S.O.P.("M1 method");}
          void m2(){     S.O.P.("M2 method");}
          void m3(){     S.O.P.("M3 method");}
          P.S.V.M(String[] args){
                    Test1  t = new Test1();
                    t.m1();
                    t.m2();
                    t.m3();
}}
```

Compiler error:

Permission reduce

Solution

```
public void m1()
{
}
```

Will it compile or not?

```
Interface Item{
            void m1();
            void m2();
            void m3();
}
class Test1 implements Item{
            public void m1()  { S.O.P.("M1 method");}
}
class Test2 extends Test2{
         public void m2()   {  S.O.P.("M2 method");}
}
class Test3 extends Test2{
            public void m3()   { S.O.P.("M3 method");}
            P.S.V.M(String[] args){
                        Test3  t = new Test3();
                        t.m1();
                        t.m2();
                        t.m3();
}}
```

Compiler error:

Solution-
Declare class as abstract class

What is the output here?

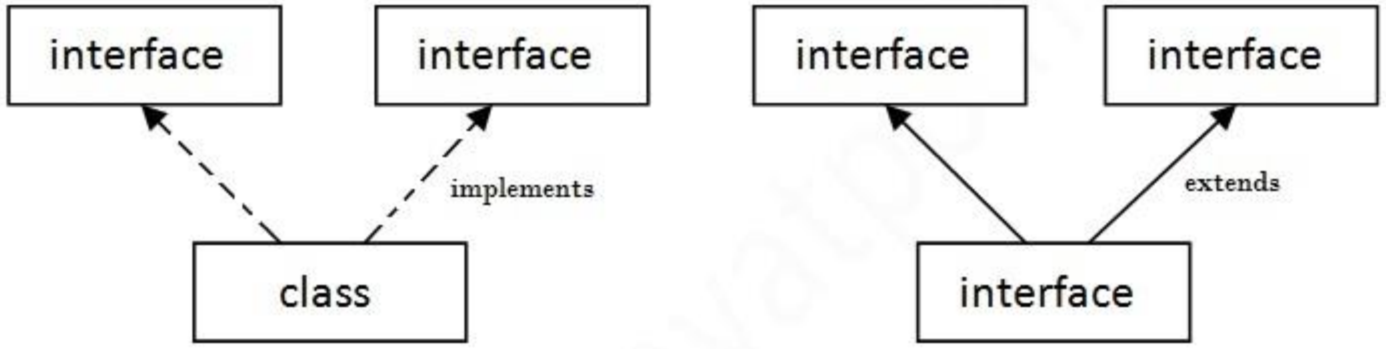
```
interface printable{
	void print();
}
class CEIT implements printable{
	public void print(){
		System.out.println("Hello Interface");
	}
	public static void main(String args[]){
	CEIT obj = new CEIT();
	obj.print();
	}
}
```

What is the output here?

```
interface Bank{
        float rateOfInterest();
}
class SBI implements Bank{
        public float rateOfInterest(){return 9.15f;}
}
class PNB implements Bank{
        public float rateOfInterest(){return 9.7f;}
}
class TestInterface{
        public static void main(String[] args){
        Bank b=new SBI();
        System.out.println("ROI: "+b.rateOfInterest());
}}
```

YES

How to achieve **Multiple inheritance** in Java by interface?



Multiple Inheritance in Java

What is the output here?

```
interface Printable{
          void print();
}
interface Showable{
          void show();
}

class CEIT implements Printable,Showable{
          public void print(){      System.out.println("Hello");}
          public void show(){       System.out.println("Welcome");}

          public static void main(String args[]){
          CEIT obj = new CEIT();
          obj.print();
          obj.show();
 }
}
```

Will it compile or not?

```
interface Printable{
        void print();
}
interface Showable{
        void print();
}

class TestMultiInterface implements Printable, Showable{
        public void print(){System.out.println("Hello");}

        public static void main(String args[]){
        TestMultiInterface obj = new TestMultiInterface();
        obj.print();
 } }
```

Is there any ambiguity in previous code?

Printable and **Showable** interface have same methods but its implementation is provided by class **TestMultiInterface**, so there is

no ambiguity.

Can we inherit interface?

one interface **extends** another interface

YES

Example

```
interface Printable{
        void print();
}
interface Showable extends Printable{
        void show();
}
class ExtendInterface implements Showable{

        public void print(){        System.out.println("Hello");}
        public void show(){         System.out.println("Welcome");}

        public static void main(String args[]){
                ExtendInterface obj = new ExtendInterface();
                obj.print();
                obj.show();
 } }
```

Java 8: Default Method in Interface

Since Java 8, we can have **method body** in interface. But we need to make it **default** method.

Example

```
interface Drawable{
        void draw();
        default void msg(){
        System.out.println("default method");}
}
class Rectangle implements Drawable{
        public void draw(){
        System.out.println("drawing rectangle");}
}
class TestInterfaceDefault{
        public static void main(String args[]){
        Drawable d=new Rectangle();
        d.draw();
        d.msg();
}}
```

Java 8: Static Method in Interface

- These static method will act as helper methods.
- These methods are the parts of interface not belongs to implementation class objects.

Example

```
interface StaticInterface{
 static void print(String str){
 System.out.println("Static method of interface:"+str);
 }
}
                class Demo implements StaticInterface{
                 public static void main(String[] args){
                   StaticInterface.print("Java 8")
                 }
                 }
```

Inner Class or nested inner class

Example (nested inner class)

```
class Outer {
        class Inner {
        public void show() {
                System.out.println("In a nested class method");
    }
  }
}
class Main {
        public static void main(String[] args) {
        Outer.Inner in = new Outer().new Inner();
        in.show();
  }
}
```

Will it compile or not?

```
class Outer {
  void outerMethod() {
    System.out.println("inside outerMethod");
  }
  class Inner {
    public static void main(String[] args){
      System.out.println("inside inner class Method");
    }
  }
}
```

Error illegal static declaration in inner class Outer.Inner public static void main(String[] args) modifier ‘static’ is only allowed in constant variable declaration

So we can’t have **static method** in a **nested inner class**

because an **inner class** is implicitly associated with an object of its **outer class** so it cannot define any **static method** for itself.

Homework

Explore three other type of inner classes by your own

2) Method Local inner classes
3) Anonymous inner classes
4) Static nested classes

2. Method Local inner classes

```
public class Outer {
        private String x = "outer";

        public void doStuff() {
                class MyInner {
                        public void seeOuter() {
                                System.out.println("x is " + x);
                        }
                }
                MyInner i = new MyInner();
                i.seeOuter();
        }
        public static void main(String[] args) {
                Outer o = new Outer();
                o.doStuff();
        }}
```

```
x is outer
```

2. Method Local inner classes

```
public class Outer {
        private static String x = “static outer";

        public void doStuff() {
                class MyInner {
                        public void seeOuter() {
                                System.out.println("x is " + x);
                        }
                }
                MyInner i = new MyInner();
                i.seeOuter();
        }
        public static void main(String[] args) {
                Outer o = new Outer();
                o.doStuff();
        }}
```

```
x is static outer
```

3. Anonymous inner classes

- A class that have no name is known as anonymous inner class in java. It should be used if you have to override method of class or interface.
- Java Anonymous inner class can be created by two ways:

1. Class (may be abstract or concrete).
2. Interface

Anonymous inner class example using abstract class

```
abstract class Person{
  abstract void eat();
}
class TestAnonymousInner{
 public static void main(String args[]){
  Person p=new Person(){
  void eat(){System.out.println("nice fruits");}
  };
  p.eat();
 }
}
```

How many class file will be generated after compilation?

1. Person.class
2. TestAnonymousInner$1.class
3. TestAnonymousInner.class

We can execute code using only TestAnonymousInner.class

Internal working of given code

```
Person p=new Person(){
void eat(){System.out.println("nice fruits");}
};
```

1. A class is created but its name is decided by the compiler which extends the Person class and provides the implementation of the eat() method.

2. An object of Anonymous class is created that is referred by p reference variable of Person type.

Conclusion

- overloading methods of a **abstract class** or interface, without having to actually subclass a **class**.

Internal class generated by the compiler

```
static class TestAnonymousInner$1 extends Person
{
  TestAnonymousInner$1(){}
  void eat()
   {
      System.out.println("nice fruits");
   }
}
```

Java anonymous inner class example using interface

```
interface Eatable{
 void eat();
}
class TestAnnonymousInner1{
 public static void main(String args[]){
 Eatable e=new Eatable(){
  public void eat(){System.out.println("nice fruits");}
 };
 e.eat();
 }
}
```

This is frequently used when you add an action listener to a widget in a GUI application.

```
button.addActionListener(new ActionListener(){
    public void actionPerformed(ActionEvent e){
        comp.setText("Button has been clicked");
    }
});
```

4. Static Nested Classes

```
class Outer {
        static class Inner {
                void go() {
                        System.out.println("Inner class reference is: " + this);
                }
        }
}

public class Test {
        public static void main(String[] args) {
                Outer.Inner n = new Outer.Inner();
                n.go();
        }
}
```

```
Inner class reference is: Outer$Inner@19e7ce87
```

Nested Interface

```
interface printable{
        void print();
        interface MessagePrintable{
                void msg();
 }
}
```

Example

```
interface Showable{
            void show();
             interface Message{
                        void msg();
  }
}
class TestNestedInterface implements Showable.Message{
            public void msg(){
                        System.out.println("Hello nested interface");}

            public static void main(String args[]){
            Showable.Message message=new TestNestedInterface();
            message.msg();
 }  }
```

How to call **show()** method in previous example?

Solution

```
interface Showable{
            void show();
             interface Message{
                        void msg();
  }
}
class TestNestedInterface implements Showable, Showable.Message{
            public void msg(){
                        System.out.println("Hello nested interface");}
            public void show(){
                        System.out.println("SHOW Hello nested interface");}

            public static void main(String args[]){
                        Showable.Message message=new TestNestedInterface();
                        message.msg();
                        Showable m = new TestNestedInterface();
                        m.show();
 } }
```

Can we declared interface within the class?

YES

Example

```
class A{
        interface Message{
                void msg();
        }
}

class InterfaceWithinClass implements A.Message{
                public void msg(){
                System.out.println("Hello nested interface");}

                public static void main(String args[]){
                 A.Message message=new InterfaceWithinClass();
                message.msg();
 }  }
```

Is there any problem in Interface?

Example

```
Interface It1
{ void m1();
  void m2();
  void m3();
  void m4();
  void m5();
  void m6();
  void m7();
  void m8();
  void m9();
  void m10();
}
```

```
Class X implements It1
{ void m1(){}
  void m2(){}
  void m3(){}
  void m4(){}
}
```

Solution

Adapter Class

Example

```
Interface It1
{ void m1();
  void m2();
  void m3();
  void m4();
  void m5();
  void m6();
  void m7();
  void m8();
  void m9();
  void m10();
}
```

```
Class X implements It1
{ void m1(){}
  void m2(){}
  void m3(){}
  void m4(){}
  void m5(){}
  void m6(){}
  void m7(){}
  void m8(){}
  void m9(){}
  void m10(){}
}
```

```
Class Test extends X{
}
```

Can Interface have constructor?

NO

Example

```
public interface sample
{
        int a=10;
        sample(){
          }
}
```

Error description:
Interfaces cannot have constructors.

Reason

- Interfaces in Java don’t have constructor because all data members in interfaces are public static final by default, they are constants(assign values at the time of declaration) .There are no data members in interfaces to initialize them through constructor.

interface doesn't allow us to create constructor.

Any Question???

Q-1 What is the output for the below code ?

```
public interface InfA {
          protected String getName();
}
public class Test implements InfA{
          public String getName(){
          return "test-name";
          }
          public static void main (String[] args){
            Test t = new Test();
            System.out.println(t.getName());
}
}
```

Options are

A. test-name

B. Compilation fails due to an error on lines 2

C. Compilation fails due to an error on lines 1

D. Compilation succeed but Runtime Exception

B. Compilation fails due to an error on lines 2

Q.02

```
class A
{
    A(String s){    }
   A(){            }
}
class B extends A{
     B(){    }
     B(String s){
     super(s);
 }
 void test()
{
 7. // insert code here
 }
 }
```

Which of the below code can be insert at line 7 to make clean compilation ?

A. A a = new B();

B. A a = new B(5);

C. A a = new A(String s);

D. All of the above

E. None of these

A. A a = new B();

```
class Alpha {
String getType() {
return "alpha";
}
}
class Beta extends Alpha {
String getType() {
return "beta";
}
}
public class Gamma extends Beta {

String getType() {
return "gamma";
}
public static void main(String[] args) {
Gamma g1 = new Alpha();
Gamma g2 = new Beta();
System.out.println(g1.getType() + " “+
g2.getType());
}
}
```

Q.03

What is the result?

A) alpha beta

B) beta beta

C) gamma gamma

D) Compilation fails.

D) Compilation fails

Q-04

```
1. public interface Traceable {
2. public static int MAX_TRACE;
3. public void trace();
4. }
5.
6. class Picture implements Traceable {
7. public void trace() {
8. System.out.println("Tracing a picture");
9. }
10. }
```

C. Compiler error on line 2

Options are-

A. Two bytecode files: Traceable.class and Picture.class

B. One bytecode file: Traceable.class

C. Compiler error on line 2

D. Compiler error on line 3

E. Compiler error on line 6

F. Compiler error on line 7

```
1. public class Outer {
2. private int x = 5;
3.
4. protected class Inner {
5. public static int x = 10;
6.
7. public void go() {
8. System.out.println(x);
9. }
10. }
11.}
```

Given the following code:

```
15. Outer out = new Outer();
16. Outer.Inner in = out.new Inner();
17. in.go();
```

Q-05

Options are-

A. The output is 10.

B. The output is 5.

C. Line 16 generates a compiler error.

D. Line 5 generates a compiler error.

D. Line 5 generates a compiler error.

Q-06

```
//Readable.java
public interface Readable {
public abstract void read();
}
//SpellCheck.java
public interface SpellCheck extends Readable {
public void checkSpelling();
}
```

Options are- Which one is true

A. The SpellCheck interface does not compile.

B. A class that implements Readable must override the read method.

C. A class that implements SpellCheck inherits both the checkSpelling and read methods.

D. A class that implements SpellCheck only inherits the checkSpelling method.

Ans-C

Q-07

```
class Parent {
    public float computePay(double d) {
        System.out.println("In Parent");
        return 0.0F;
    }
}

public class Child extends Parent {
    public double computePay(double d) {
        System.out.println("In Child");
        return 0.0;
    }

    public static void main(String [] args) {
        new Child().computePay(0.0);
    }
}
```

A. `In Parent`

B. `In Child`

C. `0.0`

D. `null`

E. The code does not compile.

Ans- E

Q-08

```java
1.  class Parent {
2.    public void print(double d) {
3.        System.out.print("Parent");
4.    }
5.  }
6.
7.  class Child extends Parent {
8.    public void print(int i) {
9.        System.out.print("Child");
10.   }
11.}
```

what is the result of the following code?

```java
15. Child child = new Child();
16. child.print(10);
17. child.print(3.14);
```

Ans-A

A. `ChildParent`

B. `ChildChild`

C. `ParentParent`

D. Line 8 generates a compiler error.

E. Line 17 generates a compiler error.